한국어 기계독해

- 1팀: NLP 이렇게 하는거조 –

신혜인, 구교선, 김동현, 신선미, 유도형, 전은정, 조상현, 하영현

한국어 기계독해

목차

1. 프로젝트 개요

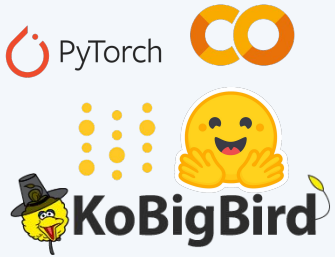
한국어 기계독해 모델 구현하기

구현 내용

한국어 사전학습 모델을 이용해서, 지문을 보고 질문에 대한 답을 주어진 문맥에서 찾아내기(MRC)

활용 툴

PyTorch
Hugging Face Transformer
KoBigBird Model
KoELECTRA Model
WandB
Colab Pro Plus



목표

- 본문에서 정확한 답변의 위치를 찾아 출력하기
- 오답은 최대한 회피하기
- 되도록 짧은 답을 산출하기

→ 평가산식인 편집거리에서 최대한 낮은 점수를 받기!

2. 팀 구성 및 역할

역할

신혜인	데일리 미팅일지 작성, KoBigBird
구교선	사전학습 모델 탐색 및 적용, 앙상블
김동현	baseline코드 정리, 데이터 전처리 및 후처리
신선미	사전학습 모델 탐색, KoELECTRA, KoBigBird
유도형	baseline코드 정리, EDA, 데이터 전처리 및 후처리, 실험 설계
전은정	사전학습 모델 탐색, KoELECTRA, KoBigBird
조상현	baseline코드 정리, KoBigBird 기반 다양한 실험
하영현	사전학습 모델 탐색, KoELECTRA, KoBigBird

주요 업무

베이스라인 코드 수정 및 분석

baseline 코드 분석 raw data 분석

사전학습 모델 탐색

Ko-ELECTRA-Base-v3 KoBigBird-BERT-Base

가설 수립 및 실험

모델, 전처리.후처리 카테고리 별 가설 수립, 편집거리 실험

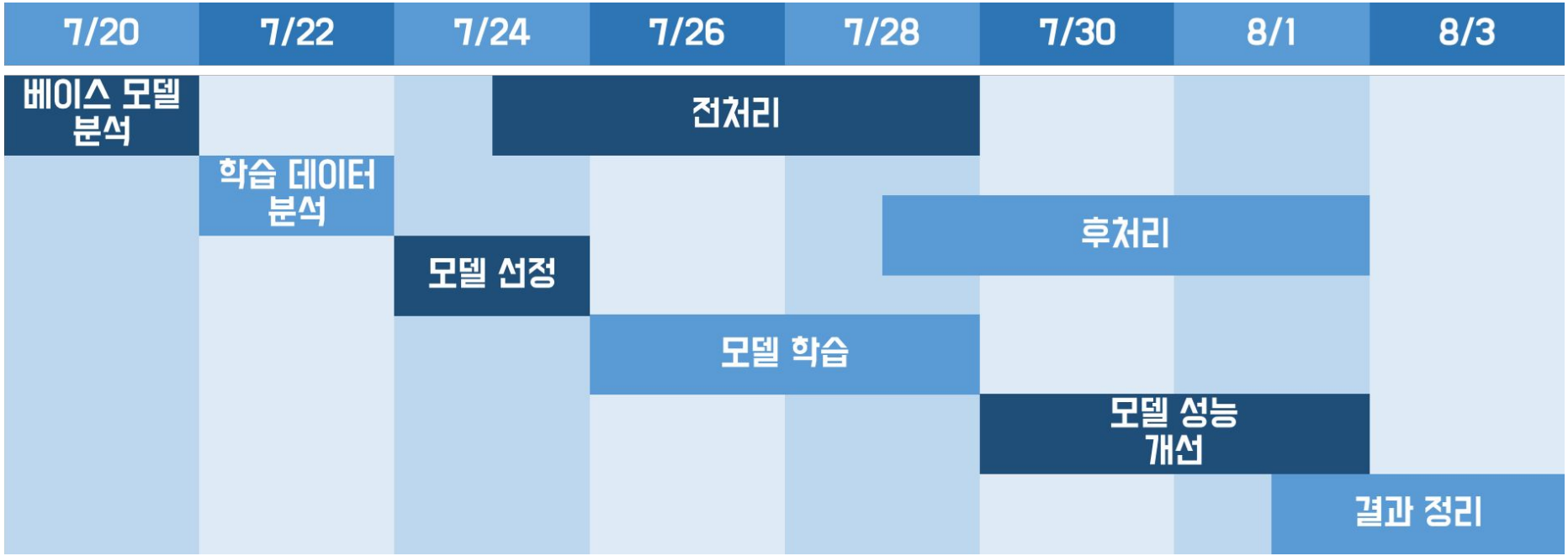
결과 비교 및 보고서 작성

선정 성능 지표 기준으로 결과 비교
비교 결과와 분석 보고서 작성

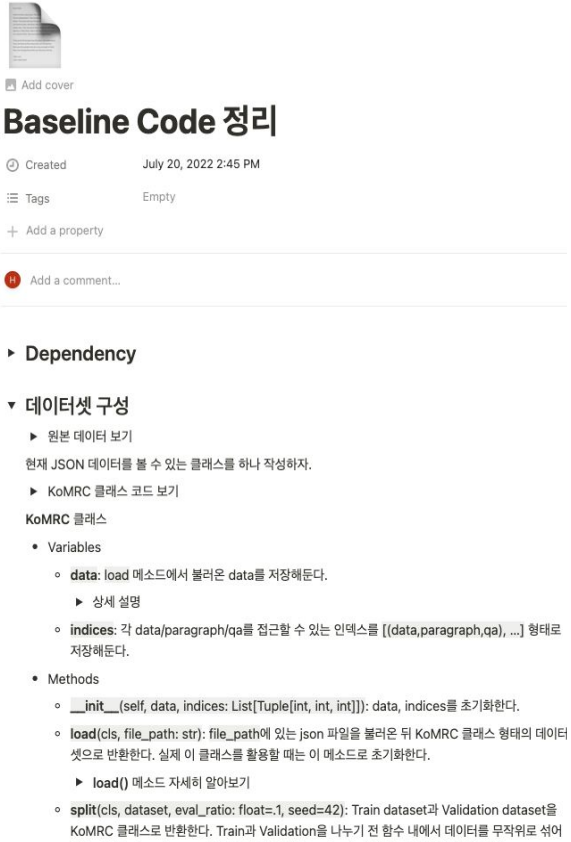
3.1 단계별 프로세스 및 일정

베이스라인 분석	후보 모델 선정	성능 비교 및 실험	평가 및 보완점 정의
베이스라인 분석을 토대로 한 전처리 및 모델 구조 파악	한국어 PLM 중 MRC에 적합한 모델 선정 및 적용	모델별, 변수별 실험 시행 및 결과 비교	프로젝트 수행 과정 자체평가 진행 및 보완점 정의
베이스라인 코드 분석 및 재작성 제공된 베이스라인 코드 분석, 프로젝트 목적에 부합하는 베이스라인 코드 재작성	**후보 모델 정의** **KoELECTRA** **KoBigBird**	**성능 지표 선정** **Kaggle score** **Valid Score** **편집 거리(self test)**	**자체평가** 프로젝트 수행 결과를 통한 자체평가 진행(pro / con)
EDA 제공된 train, test data 분석 및 전처리 가설 수립	**모델별 기본 hyper-parameter 탐색** 기본 하이퍼 파라미터 탐색	**Best Model 선정** 편집거리(self test), valid score 비교해 1차 선정 Kaggle score 로 검증	**보완점 정의** 프로젝트 수행 시 부족했던 보완점 정의

3.1 단계별 프로세스 및 일정



3.2 베이스 라인 분석



Add cover

Baseline Code 정리

Created July 20, 2022 2:45 PM

Tags Empty

Add a property

Add a comment...

▸ **Dependency**

▾ **데이터셋 구성**

▸ 원본 데이터 보기

현재 JSON 데이터를 볼 수 있는 클래스를 하나 작성하자.

▸ KoMRC 클래스 코드 보기

KoMRC 클래스

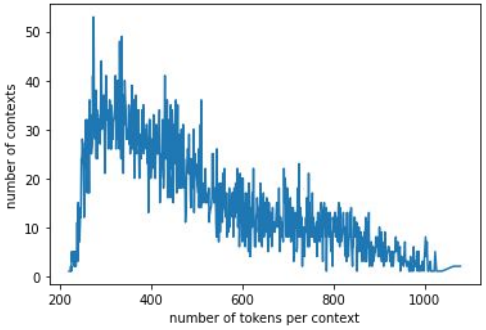
- Variables
  - **data**: load 메소드에서 불러온 data를 저장해둔다.
    - ▸ 상세 설명
  - **indices**: 각 data/paragraph/qa를 접근할 수 있는 인덱스를 [(data,paragraph,qa), ...] 형태로 저장해둔다.
- Methods
  - **__init__**(self, data, indices: List[Tuple[int, int, int]]): data, indices를 초기화한다.
  - **load**(cls, file_path: str): file_path에 있는 json 파일을 불러온 뒤 KoMRC 클래스 형태의 데이터셋으로 반환한다. 실제 이 클래스를 활용할 때는 이 메소드로 초기화한다.
    - ▸ **load()** 메소드 자세히 알아보기
  - **split**(cls, dataset, eval_ratio: float=.1, seed=42): Train dataset과 Validation dataset을 KoMRC 클래스로 반환한다. Train과 Validation을 나누기 전 함수 내에서 데이터를 무작위로 섞어

```python
import os
from statistics import mean

import torch.nn.functional as F
from torch.nn.utils import clip_grad_norm_

os.makedirs('dump', exist_ok=True)
train_losses = []
dev_losses = []

step = 0

for epoch in range(1, 31):
    print("Epoch", epoch)
    # Training
    running_loss = 0.
    losses = []
    progress_bar = tqdm(train_loader, desc='Train')
    for batch in progress_bar:
        del batch['guid'], batch['context'], batch['question'], batch['position
        # 학습에서 굳이 쓰지 않는 batch 내 데이터들을 제거한다.
        # batch = {key: value.cuda() for key, value in batch.items()}
        start = batch.pop('start')
        end = batch.pop('end')

        start_logits, end_logits = model(**batch)
        # batch 앞에 asterisk 2개를 붙여 batch를 unpacking하여 보내준다.
        loss = F.cross_entropy(start_logits, start) + F.cross_entropy(end_logit
        (loss / accumulation).backward()
        running_loss += loss.item()
        del batch, start, end, start_logits, end_logits, loss
        # loss 값을 구하고 backward가 끝났으니 잠시 사용했던 값들을 날려보냄

        step += 1
        if step % accumulation:
            continue
        # 미리 정해둔 accumulation 주기가 아니면 기울기를 계속 쌓아나갈 것이다.

        clip_grad_norm_(model.parameters(), max_norm=1.)
        # Gradient Exploding을 방지하기 위한 Gradient Clipping
        optimizer.step()
        optimizer.zero_grad(set_to_none=True)
        # Gradient 초기화

        losses.append(running_loss / accumulation)
        running_loss = 0.
        progress_bar.set_description(f"Train - Loss: {losses[-1]:.3f}")
    train_losses.append(mean(losses))
```

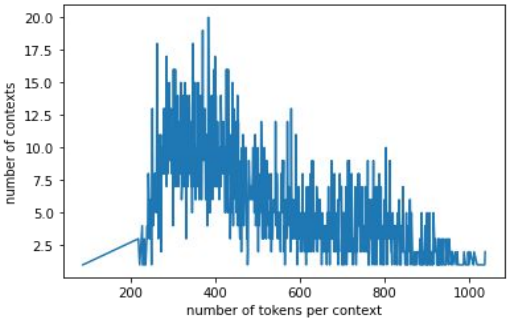
- 베이스라인 작동 방식 분석
- 모델의 기본 구조 파악
- 사용 라이브러리 관련 유의사항 파악
- 사용 클래스 및 파라미터 파악

Raw Data 탐색

Raw Data	Min Token	Max Token
Train.json	221	1078
Test.json	87	1037



Train 데이터 (12037 samples)
평균 496.32 tokens



Test 데이터 (4008 samples)
평균 501.51 tokens

한국경제

한국경제신문 (57.35%)

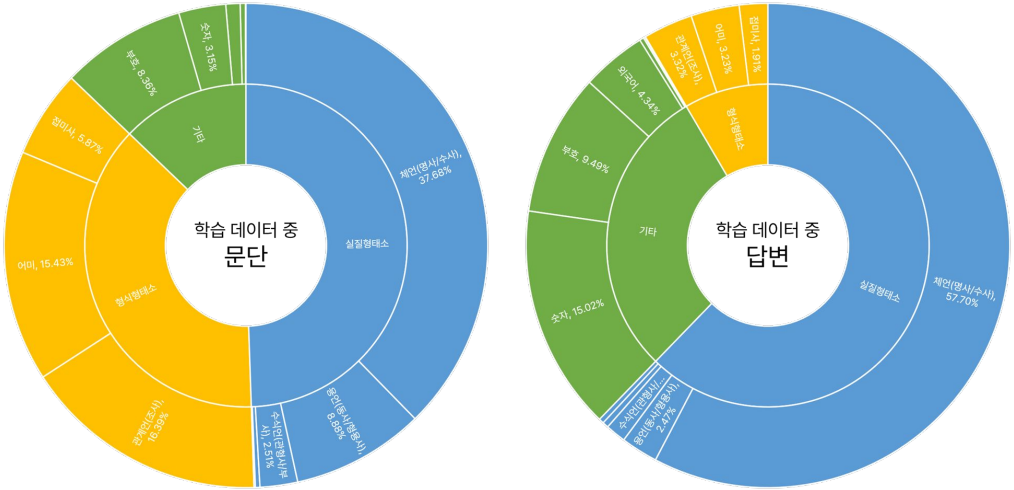
WIKIPEDIA
The Free Encyclopedia

위키피디아 (29.50%)

acrofan

아크로팬 (13.15%)

Raw Data 탐색



- 형식 형태소를 지우면 전체 문단 길이를 51.54% 가량 단축하는 효과를 기대할 수 있음
- 답변은 문단에 비해 실질 형태소와 기타(숫자, 부호, 외국어 등) 형태소의 비중이 높음

모델 별 max length 탐색

Model	max length
KoELECTRA-Base-v3	512
KoELECTRA-small-v3	512
BERT	512
Longformer	4096
KoBigBird-BERT-Base	4096

KoELECTRA-Base-v3

학습데이터 뉴스, 위키, 나무위키, 신문, 구어, 메신저, 웹

URL https://github.com/monologg/KoELECTRA

Add a comment...

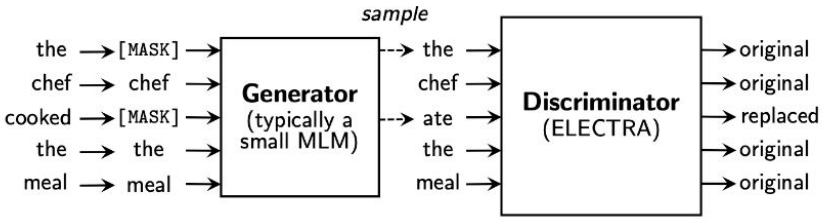
KLEU 사용 아님

모델 개요

- ELECTRA를 기반으로 한국어 텍스트를 학습한 모델
- `Transformers` 라이브러리를 통해 적용할 수 있음
- v3 학습 데이터셋은 v1과 v2에서 학습한 뉴스, 위키, 나무위키에 더해 모두의 말뭉치를 이용하여 신문, 문어, 구어, 메신저, 웹을 추가로 학습
  - 따라서 모델의 크기가 매우 크기 때문에 `base-v3` 보다 `small-v3` 을 사용하는 것이 자원 할당 측면에서 더 긍정적
    - Base Model : 431M / Small Model: 54M
- task별 finetuning한 결과에서 v3가 한국어 위키 문답 데이터셋인 **KorQuaD**에서 타 모델 대비 높은 성능을 나타냄. (첨부 그림 참조)

Base Model

	Size	NSMC (acc)	Naver NER (F1)	PAWS (acc)	KorNLI (acc)	KorSTS (spearman)	Question Pair (acc)	KorQuaD (Dev) (EM/F1)	Korean-Hate-Speech (Dev) (F1)
KoBERT	351M	89.59	87.92	81.25	79.62	81.59	94.85	51.75 / 79.15	66.21
XLM-Roberta-Base	1.03G	89.03	86.65	82.80	80.23	78.45	93.80	64.70 / 88.94	64.06
HanBERT	614M	90.06	87.70	82.95	80.32	82.73	94.72	78.74 / 92.02	**68.32**
KoELECTRA-Base	423M	90.33	87.18	81.70	80.64	82.00	93.54	60.86 / 89.28	66.09
KoELECTRA-Base-v2	423M	89.56	87.16	80.70	80.72	82.30	94.85	84.01 / 92.40	67.45
KoELECTRA-Base-v3	431M	**90.63**	**88.11**	**84.45**	**82.24**	**85.53**	**95.25**	**84.83 / 93.45**	67.61



- **KoELECTRA-Base-v3 모델**
  - ELECTRA 기반 모델, 34GB 한국어 text 사전학습된 모델
  - Replaced Token Detection 방식 학습
  - BERT 보다 더 좋은 성능
  - BERT와 같은 512의 max seq length를 가짐

ko-BigBird-BERT-Base

학습데이터 모두의 말뭉치, 한국어 위키, Common Crawl, 뉴스 데이터 등 다양한 데이터로 학습

URL https://github.com/monologg/KoBigBird

Add a comment...

1 resolved comment

KLEU 사용 아님

모델 개요

- BigBird: Transformers for Longer Sequences에서 소개된 **sparse-attention** 기반의 모델로, 일반적인 BERT보다 **더 긴 sequence**를 다룰 수 있습니다.

  Longer Sequence - 최대 512개의 token을 다룰 수 있는 BERT의 8배인 **최대 4096개의 token**을 다룸

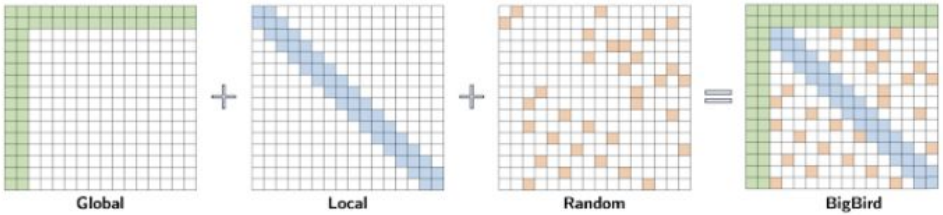
  Computational Efficiency - Full attention이 아닌 **Sparse Attention**을 이용하여 O(n2)에서 **O(n)**으로 개선

- 높은성능

	NSMC (acc)	KLUE-NLI (acc)	KLUE-STS (pearsonr)	Korquad 1.0 (em/f1)	KLUE MRC (em/rouge-w)
KoELECTRA-Base-v3	91.13	86.87	**93.14**	85.66 / 93.94	59.54 / 65.64
KLUE-RoBERTa-Base	91.16	86.30	92.91	85.35 / 94.53	69.56 / 74.64
KoBigBird-BERT-Base	**91.18**	**87.17**	92.61	**87.08 / 94.71**	**70.33 / 75.34**

- `KLUE`, `Korquad 1.0` 모두 **dev set**으로 평가
- `KoELECTRA-Base-v3` 와 `KLUE-RoBERTa-Base` 의 KLUE dataset 관련 점수는 KLUE Paper의 `A. Dev Set Results` 에서 참고

- 모두의 말뭉치, 한국어 위키, Common Crawl, 뉴스 데이터 등 다양한 데이터로 학습
- `Transformers` 라이브러리를 통해 적용할 수 있음
- Huggingface Hub에 업로드된 모델을 곧바로 사용 가능
- **BigBirdTokenizer** 대신에 `BertTokenizer` 를 사용(`AutoTokenizer` 사용시 `BertTokenizer` 가 로



- **KoBigBird-BERT-Base 모델**
  - Transformers for longer sequences 에서 소개된 sparse-attention 기반의 모델
  - 계산 효율성 개선
  - BERT 보다 더 긴 4096 max seq length를 가짐

4.1.1 전처리

1. 형태소 선택적 적용

MeCab을 이용해 문장들을 토큰화 할때 형식형태소를 제외하여 실질 형태소와 한글 이외의 토큰(외국어, 한자, 숫자)만 취급하여 학습

<실질 형태소 목록>

NNG	일반 명사
NNP	고유 명사
NNB	의존 명사
NNBC	단위를 나타내는 명사
NR	수사
NP	대명사
VV	동사
VA	형용사
VX	보조 용언
VCP	긍정 지정사
VCN	부정 지정사
MM	관형사
MAG	일반 부사
MAJ	접속 부사
IC	감탄사

<기타 형태소 목록>

XR	어근
SF	마침표, 물음표, 느낌표
SE	줄임표 …
SSO	여는 괄호 (, [
SSC	닫는 괄호),]
SC	구분자 , · / :
SY	붙임표(물결,숨김,빠짐), 기타 기호(논리수학기호, 화폐기호)
SL	외국어
SH	한자
SN	숫자

<Problems>

- 형태소 분석기의 정확도가 다소 떨어져 고유 명사가 잘려나가는 문제가 발생
- KoELECTRA, KoBigBird 등의 모델의 토크나이저 호환성 문제

→ **적용하지 않기로 결정**

4.1.1 전처리

2. 최단 길이의 답만 선택적 학습

Answer가 여러 개인 경우, 가장 짧은 길이의 Answer만 학습

```
for answer in dataset[0]['answers']:
    print(answer['text'])
else:
    print(len(dataset[0]['answers']))
```

```
한 달가량
한 달
2
```

```
for answer in dataset[0]['answers']:
    print(answer['text'])
else:
    print(len(dataset[0]['answers']))
```

```
한 달
1
```

<Result>

Kaggle score 2.74 → 2.64로 약 0.1 점 가량 상승

4.1.1 전처리

3. 토큰화 된 형태소에서 “##” 제거

토큰화 할 때 단어 단위 분절 과정에서 생성되는 “##” 삭제

```
print(tokenizer.tokenize('올여름 장마가 17일 제주도에서 시작됐다.'))
```

```
['올', '##여름', '장마', '##가', '17', '##일', '제주도', '##에', '##서', '시작', '##됐', '##다', '.']
```

```
for token in tokenizer.tokenize('올여름 장마가 17일 제주도에서 시작됐다.'):
    morph.append(token[2:] if token.startswith("##") else token)
print(morph)
```

```
['올', '여름', '장마', '가', '17', '일', '제주도', '에', '서', '시작', '됐', '다', '.']
```

4.1.1 전처리

4. AI-Hub 학습 데이터 추가

1

AI-Hub에서 비슷한 도메인을 가진
뉴스 기사 기계독해 데이터 train 데이터 중
extraction 데이터셋을 선택

2

Answer의 길이가 **20글자 이하**인 데이터만 추출
(기존의 train 데이터의 95%가 20글자)

3

기존의 train 데이터의 개수: **12,037**

AI-Hub 학습 데이터의 개수* : 198,748

*전체 데이터 중 답안 길이가 20글자 미만인 데이터

4

AI-Hub 학습 데이터 수정
(question 끝자리에 '?' 추가)

4.1.1 전처리

4. AI-Hub 학습 데이터 추가

4

AI-Hub 학습 데이터 수정
(question 끝자리에 '?' 추가)

추가된 데이터의 question 끝 자리에
'?' 문자를 추가해서 데이터의 형태를 일관성있게 통일

```
print("기존 데이터 : ", dataset1[0]['question'])
print("추가 데이터 : ", dataset2[0]['question'])
```

```
기존 데이터 :  북태평양 기단과 오호츠크해 기단이 만나 국내에 머무르는 기간은?
추가 데이터 :  익산시에서 시민들의 건강을 위해 운영을 준비 중인 시설은 뭐야
```

- 기존 데이터의 question은 '?'로 끝나는 반면, 추가된 데이터의 question엔 '?'가 없다.

4.1.1 전처리

4. AI-Hub 학습 데이터 추가

198,748 중 랜덤으로 **7000개** 가량 추출하여 추가

- KoMRC 클래스에 extraction 함수를 정의
- 추가 데이터셋에서 랜덤으로 정해진 숫자만큼을 추출
- 추출된 데이터를 기존 train.json 데이터셋에 추가

```python
class KoMRC :

  @classmethod
  def extraction(cls, dataset, data_number: int, save=False):
      '''
      첫 번째 인자 : KoMRC의 인스턴스
      두 번째 인자 : 추출할 data 개수 (단, data단위로 추출, indices단위가 아님)
      반환값 : KoMRC의 인스턴스
      '''
      if data_number > len(dataset):
          raise Exception("입력한 인자가 데이터셋 크기보다 큽니다.")
      else:
          data = copy.deepcopy(dataset._data)
          data['data'] = random.sample(data['data'], data_number)
          indices = []
          for d_id, document in enumerate(data['data']):
              for p_id, paragraph in enumerate(document['paragraphs']):
                  for q_id, _ in enumerate(paragraph['qas']):
                      indices.append((d_id, p_id, q_id))

          # 잘라낸 데이터를 저장
          if save:
              with open("cut_result.json", "w") as write_file:
                  json.dump(data, write_file, indent=4)

      return cls(data, indices)
```

4.1.2 모델

KoELECTRA

1. **토크나이저 변경(MeCab -> ElectraTokenizer)**
   - 기존 baseline의 Mecab 토크나이저로 토큰화가 제대로 되지 않는 것을 확인한 후 ElectraTokenizer로 변경.

2. **doc_stride 설정**
   - 지문의 길이가 KoELECTRA의 max_seq_length보다 길기 때문에 적용 시도.
   - doc_stride 길이 64, 128, 256 변화.

3. **small, base 모델 비교**
   - 편집거리를 통한 결과 비교.

4.1.2 모델

KoELECTRA

1. 토크나이저 변경

- MeCab → ElectraTokenizer 사용
- 토크나이저 변경에 따라 TokenizedKoMRC 클래스를 수정하여 적용

▾ 모델 별 토크나이저 분석

MeCab

BERT 자체는 사전학습 모델이 아니기 때문에, 어떠한 토크나이저를 써도 상관없다. 다만 Baseline 코드에서는 KoNLPy의 MeCab 토크나이저를 사용하고 있다.

MeCab v2.0에서의 품사 태그는 여기서 구글 스프레드 시트로 확인할 수 있다. 토큰 구조는 표층형/품사태그의미부류로 구성되어 있다.

MeCab은 형태소를 쪼갤 때 음운 단위에서 더 쪼개지 않는 특성을 지닌다.

KoELECTRA - Wordpiece

KoELECTRA에서는 추가적인 tokenization 파일을 만들지 않기 위하여 Huggingface의 Tokenizers 라이브러리를 이용하여 토크나이저를 구축했다. ELECTRA에서 사용하는 Wordpiece는 sub-word based tokenizer로, 위에서 봤던 토크나이저와 유사하게 형태소 단위로 쪼개준다.

```
13143  김포
13144  이번에
13145  ##내기
13146  배터지
13147  달려
13148  ##습니까
13149  ##마른
13150  검색
13151  제일
13152  ##강설
13153  ##해오므
13154  ##일은
13155  쓸려
13156  그것이
13157  ##되더니
13158  ##해에도
13159  ##생
13160  형사
13161  유명한
13162  정규
13163  불러
13164  ##표##
13165  제3
```

KoELECTRA Tokenizer에는 약 3만개의 단어들이 위와 같은 형식으로 저장되어 있고, 뒤에 200개 정도 [unused] 토큰을 붙여두었다.

```
>>> from transformers import ElectraTokenizer
>>> tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-base-v3-di
>>> tokenizer.tokenize("[CLS] 한국어 ELECTRA를 공유합니다. [SEP]")
['[CLS]', '한국어', 'EL', '##EC', '##TRA', '##를', '공유', '##합니다', '.', '[SEP]']
>>> tokenizer.convert_tokens_to_ids(['[CLS]', '한국어', 'EL', '##EC', '##TRA', '#
[2, 11229, 29173, 13352, 25541, 4110, 7824, 17788, 18, 3]
```

보다시피 Wordpiece 특성상 음운을 쪼개지 않으나 의존적인 형태소들은 앞에 접두사 **##**가 붙는다. 따라서 만약 ELECTRA를 활용한다면 **TokenizedKoMRC** 클래스의 **_tokenize_with_position()** 메소드는 아래와 같이 수정해야 할 것이다. (실제로 동작하는지는 검증해봐야 함)

```
def _tokenize_with_position(sentence: str) -> List[Tuple[str, Tuple[int, int]]]
    position = 0
    tokens = []
    for morph in konlpy.tag.Mecab().morphs(sentence):
        length = len(morph)-2 if '##' in morph else len(morph)
        position = sentence.find(morph, position)
```

```
>>> from transformers import ElectraTokenizer
>>> tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-base-v3-di
>>> tokenizer.tokenize("[CLS] 한국어 ELECTRA를 공유합니다. [SEP]")
['[CLS]', '한국어', 'EL', '##EC', '##TRA', '##를', '공유', '##합니다', '.', '[SEP]']
>>> tokenizer.convert_tokens_to_ids(['[CLS]', '한국어', 'EL', '##EC', '##TRA', '#
[2, 11229, 29173, 13352, 25541, 4110, 7824, 17788, 18, 3]
```

보다시피 Wordpiece 특성상 음운을 쪼개지 않으나 의존적인 형태소들은 앞에 접두사 **##**가 붙는다. 따라서 만약 ELECTRA를 활용한다면 **TokenizedKoMRC** 클래스의 **_tokenize_with_position()** 메소드는 아래와 같이 수정해야 할 것이다. (실제로 동작하는지는 검증해봐야 함)

```
def _tokenize_with_position(sentence: str) -> List[Tuple[str, Tuple[int, int]]]
    position = 0
    tokens = []
    for morph in konlpy.tag.Mecab().morphs(sentence):
        length = len(morph)-2 if '##' in morph else len(morph)
        position = sentence.find(morph, position)
        tokens.append((morph, (position, position + length)))
        position += length
    return tokens
```

BERT & Longformer

BERT는 ELMo의 단점을 보완하여 현재 top-level의 성능을 보여주는 많은 모델들의 기초로 쓰이고 있다. 다만 BERT는 긴 input을 잘 처리하지 못하는데, Longformer는 긴 input을 처리하기 위해 고안된 모델이기에 우리의 태스크에 적합하다. 하지만 둘 다 모델 자체를 사전학습해둔 것은 아니기 때문에 KoBERT 등을 (KoLongformer는 없음) 활용하거나, 우리가 직접 토크나이저를 선택한 뒤 진행해야 한다.

KoBigBird

KoBigBird는 BERT의 각종 단점을 보완하여 BERT보다 더욱 긴 input을 다룰 수 있는 모델이다. BertTokenizer를 사용하여 진행한다.

```
7586  ##아지
7587  측구
7588  ##다름
7589  리부
7590  한테
7591  이튿
7592  2000
7593  다른
7594  창고
7595  알모
7596  안구
7597  싫컵
7598  으로부터
7599  먼축
7600  ##한다
7601  마지
7602  검사
7603  전허
7604  출을직
7605  갔기
7606  조조
7607  계번
7608  사상
```

HuggingFace Transformers와 호환되도록 만들어졌으며, BigBirdTokenizer와의 호환을 위해 bos token인 <s>, eos token인 </s>가 추가되었다. 특이점이라면 <unused> 토큰이 vocab의 앞에 와있다는 것이다. (상세사항) 이외에는 KoELECTRA에서의 설명을 참고하면 될 것이다.

4.1.2 모델

KoELECTRA

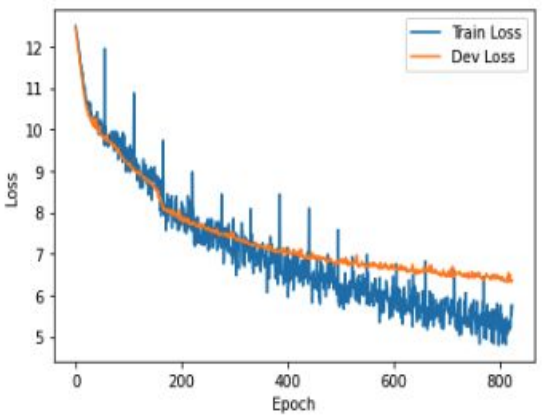
2. doc_stride 설정

- doc_stride 64, 128, 256 변화 시도 결과,
  유의미한 차이가 발견되지 않아 ELECTRA
  공식문서의 기본 하이퍼파라미터인 128 적용
  → 남아있는 답안의 개수가 절반 이하

- 데이터를 추가 + doc_stride를 같이 설정한 경우
  doc_stride가 False인 경우보다 짧은 편집거리를
  도출
  → 그러나 train, valid loss가 높고,
  답안 후처리 이후 남아있는 답안의 개수가 전체
  테스트 셋의 절반 이하인 1,817개에 불과하여
  최종적으로 채택하지 않음

구분	편집거리	train loss	valid loss	예측답/정답 길이비
True	**1.766**	5.756	6.352	0.666
False	4.657	**3.971**	**4.774**	0.888

```
doc64.isnull().sum()

Id              0
Predicted    2306
dtype: int64

doc256.isnull().sum()

Id              0
Predicted    2315
dtype: int64

doc128.isnull().sum()

Id              0
Predicted    2451
dtype: int64
```



4.1.2 모델

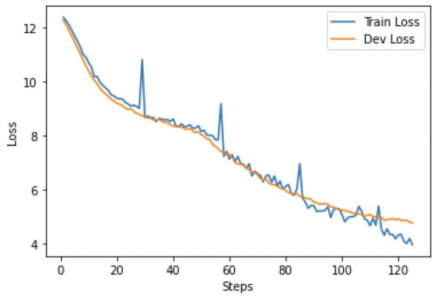
KoELECTRA

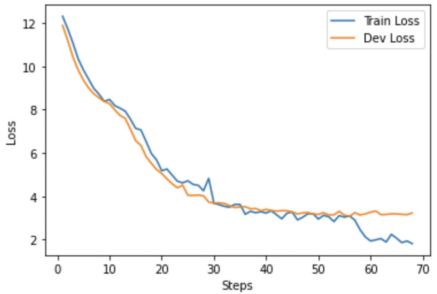
3. small, base 모델 비교

- small: 사이즈 54M
- base: 사이즈 431M

→ base의 경우 편집 거리 0 이 538개 (44.72%)
→ small의 경우 편집 거리 0이 260개(21.615)
→ 사이즈의 영향을 받아 small에서 예측답/정답 길이비가 짧고 가벼운 만큼 loss가 낮긴 하지만, base가 좀 더 성능이 좋은 것으로 확인.

구분	편집거리	train loss	valid loss	예측답/정답 길이비
small	4.657	**1.810**	**3.224**	0.888
base	**3.259**	3.971	4.774	0.912





4.1.2 모델

KoBigBird

1. **토크나이저 변경(MeCab -> BertTokenizer)**
   - 사전학습 모델 사용 시, 해당 모델 별로 권장하는 tokenizer를 사용해야 함
   - KoBigBird의 경우 권장하는 BertTokenizer 사용

2. **하이퍼 파라미터 설정**
   - learning rate, optimizer option(weight decay 등) 다양한 하이퍼 파라미터 설정
   - 성능 평가지표가 편집거리라 일정 수준의 성능에 도달하면, 하이퍼 파라미터 수정보다, 결과값에 직접적인 영향을 주는 전처리와 후처리의 영향이 더 큰것을 확인

3. **max_length 설정**
   - 최대 길이가 4096으로 긴 길이의 text를 다룰 수 있음
   - max_length 변경에 따른 성능의 차이를 확인 하기 위해 2048, 4096을 비교

4.1.2 모델

KoBigBird

1. 토크나이저 변경

- AutoTokenizer시 적용되는 BertTokenizer 사용
- 토크나이저 변경에 따라 TokenizedKoMRC 클래스를 수정하여 적용

▾ 모델 별 토크나이저 분석

MeCab

BERT 자체는 사전학습 모델이 아니기 때문에, 어떠한 토크나이저를 써도 상관없다. 다만 Baseline 코드에서는 KoNLPy의 MeCab 토크나이저를 사용하고 있다.

MeCab v2.0에서의 품사 태그는 여기서 구글 스프레드 시트로 확인할 수 있다. 토큰 구조는 표층형/품사태그/의미부류로 구성되어 있다.

MeCab은 형태소를 쪼갤 때 음운 단위에서 더 쪼개지 않는 특성을 지닌다.

KoELECTRA - Wordpiece

KoELECTRA에서는 추가적인 tokenization 파일을 만들지 않기 위하여 Huggingface의 Tokenizers 라이브러리를 이용하여 토크나이저를 구축했다. ELECTRA에서 사용하는 Wordpiece는 sub-word based tokenizer로, 위에서 봤던 토크나이저와 유사하게 형태소 단위로 쪼개준다.

```
13143  김영
13144  이번에
13145  ##세기
13146  봄은커
13147  달콤
13148  ##십니까
13149  ##마든
13150  검색
13151  제일
13152  ##관심
13153  ##해오는
13154  ##달은
13155  돌전
13156  그것이
13157  ##되어도
13158  ##튜에도
13159  ##ns
13160  철사
13161  두껍한
13162  접규
13163  뽑리
13164  ##표를
13165  계3
```

KoELECTRA Tokenizer에는 약 3만개의 단어들이 위와 같은 형식으로 저장되어 있고, 뒤에 200개 정도 [unused] 토큰을 붙여두었다.

```
>>> from transformers import ElectraTokenizer
>>> tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-base-v3-di
>>> tokenizer.tokenize("[CLS] 한국어 ELECTRA를 공유합니다. [SEP]")
['[CLS]', '한국어', 'EL', '##EC', '##TRA', '##를', '공유', '##합니다', '.', '[SEP]']
>>> tokenizer.convert_tokens_to_ids(['[CLS]', '한국어', 'EL', '##EC', '##TRA', '#
[2, 11229, 29173, 13352, 25541, 4110, 7824, 17788, 18, 3]
```

보다시피 Wordpiece 특성상 음운을 쪼개지 않으나 의존적인 형태소들은 앞에 접두사 **##**가 붙는다. 따라서 만약 ELECTRA를 활용한다면 **TokenizedKoMRC** 클래스의 **_tokenize_with_position()** 메소드는 아래와 같이 수정해야 할 것이다. (실제로 동작하는지는 검증해봐야 함)

```
def _tokenize_with_position(sentence: str) -> List[Tuple[str, Tuple[int, int]]]
    position = 0
    tokens = []
    for morph in konlpy.tag.Mecab().morphs(sentence):
        length = len(morph)-2 if '##' in morph else len(morph)
        position = sentence.find(morph, position)
```


```
>>> from transformers import ElectraTokenizer
>>> tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-base-v3-di
>>> tokenizer.tokenize("[CLS] 한국어 ELECTRA를 공유합니다. [SEP]")
['[CLS]', '한국어', 'EL', '##EC', '##TRA', '##를', '공유', '##합니다', '.', '[SEP]']
>>> tokenizer.convert_tokens_to_ids(['[CLS]', '한국어', 'EL', '##EC', '##TRA', '#
[2, 11229, 29173, 13352, 25541, 4110, 7824, 17788, 18, 3]
```

보다시피 Wordpiece 특성상 음운을 쪼개지 않으나 의존적인 형태소들은 앞에 접두사 **##**가 붙는다. 따라서 만약 ELECTRA를 활용한다면 **TokenizedKoMRC** 클래스의 **_tokenize_with_position()** 메소드는 아래와 같이 수정해야 할 것이다. (실제로 동작하는지는 검증해봐야 함)

```
def _tokenize_with_position(sentence: str) -> List[Tuple[str, Tuple[int, int]]]
    position = 0
    tokens = []
    for morph in konlpy.tag.Mecab().morphs(sentence):
        length = len(morph)-2 if '##' in morph else len(morph)
        position = sentence.find(morph, position)
```


BERT & Longformer

BERT는 ELMo의 단점을 보완하여 현재 괜찮한 모델들의 기초로 쓰이고 있다. 다만 BERT는 긴 input을 잘 처리하지 못하는데, Longformer는 긴 input을 처리하기 위해 고안된 모델이기에 우리의 태스크에 적합하다. 하지만 둘 다 모델 자체를 사전학습해둔 것은 아니기 때문에 KoBERT 등을 (KoLongformer는 없음) 활용하거나, 우리가 직접 토크나이저를 선택한 뒤 진행해야 한다.

KoBigBird

KoBigBird는 BERT의 각종 단점을 보완하여 BERT보다 더욱 긴 input을 다룰 수 있는 모델이다.
BertTokenizer를 사용하여 진행한다.

```
7586  ##이지
7587  축구
7588  ##이를
7589  피부
7590  한테
7591  아동
7592  2000
7593  다르
7594  참고
7595  음업
7596  연구
7597  ##합
7598  으로부터
7599  반복
7600  ##번다
7601  마지
7602  검사
7603  건축
7604  홈등록
7605  장기
7606  조건
7607  개번
7608  사상
```

HuggingFace Transformers와 호환되도록 만들어졌으며, BigBirdTokenizer와의 호환을 위해 bos token인 <s>, eos token인 </s>가 추가되었다. 특이점이라면 <unused> 토큰이 vocab의 앞에 와있다는 것이다. (상세사항) 이외에는 KoELECTRA에서의 설명을 참고하면 될 것이다.

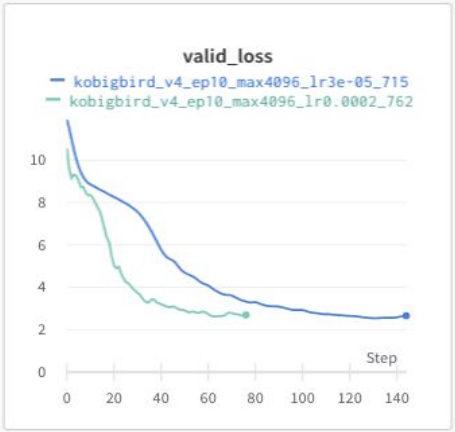
4.1.2 모델

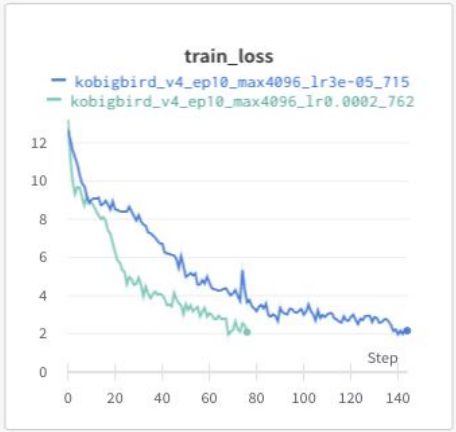
KoBigBird

2. 하이퍼 파라미터 설정

- KoBigBird 공식 문서에서 QA용 fine tuning 값을 참고하여 파라미터를 조정
- 조정 파라미터
  - learning rate 변경 (2e-4 → 3e-5)
  - weight decay 적용 (0 → 1e-5)
  - adam epsilon 적용 (0 → 1e-8)
- 큰 변화를 찾아볼 수 없었으며, 기존 파라미터 대비 학습 시간이 3배 가량 길어짐
  - 조정 전 1h 8m → 조정 후 2h 50m

구분	편집거리	train loss	valid loss	예측답/정답 길이비
조정	2.054	2.137	**2.659**	0.982
기존	**2.032**	**1.983**	2.703	0.963





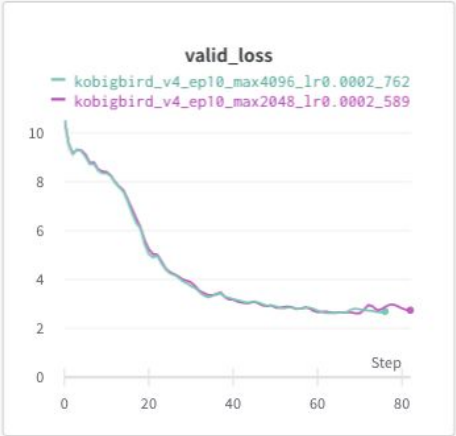
4.1.2 모델

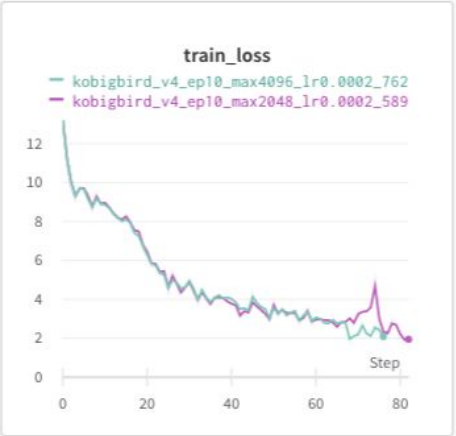
KoBigBird

3. max_length 변경 : 2048 vs 4096 비교

Max length	편집거리	train loss	valid loss	예측답/정답 길이비
2048	2.101	**1.950**	2.734	0.941
4096	**2.032**	1.983	**2.703**	0.963

- 매우 근소한 차이지만,
  최대 길이가 4096일 때 2048보다 대부분의 지표에서 더 나은 성능을 보임
- 큰 차이는 아니지만 더 많은 텍스트 인풋을 안정적으로 받기 위해 최대 길이는 4096으로 설정



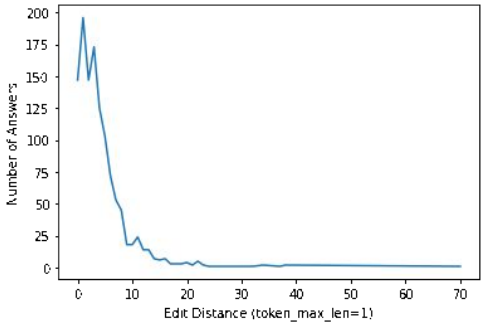


4.1.3 후처리

1. 긴 답변의 경우 답안 삭제

8 token 초과할 경우 답안 삭제

```
count    14397.000000
mean         6.040633
std          5.485509
min          1.000000
50%          5.000000
75%          7.000000
80%          8.000000
85%          9.000000
90%         11.000000
95%         15.000000
96%         17.000000
97%         19.000000
98%         22.000000
99%         28.000000
max        100.000000
Name: length, dtype: float64
```



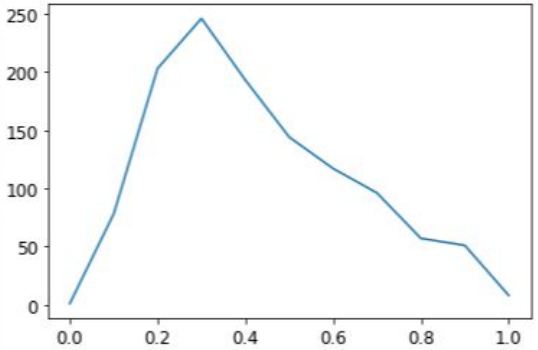
- train data의 answer 길이 분석
- 95% 이하의 data 가 15자 이내
- 일반적으로 token의 경우 1~2 글자

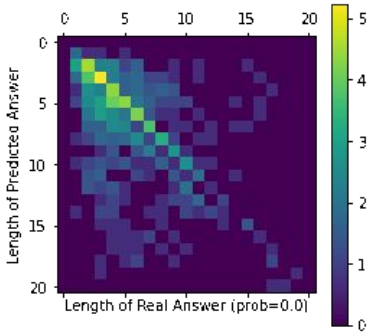
→ **8 token 초과할 경우 삭제**

4.1.3 후처리

2. 낮은 확률의 경우 삭제

답안 확률 30% 이내의 경우 삭제





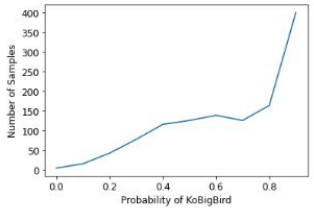
- KoBigBird data의 answer 확률 분석
- 25~35% 구간에 답안 다수 존재
- 낮은 확률의 경우 오답의 확률 증가

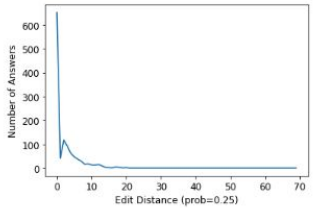
→ 확률 30% 이내 경우 삭제

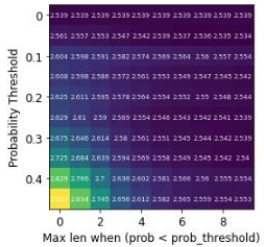
4.1.3 후처리

3. Token 길이와 확률별 삭제 실험

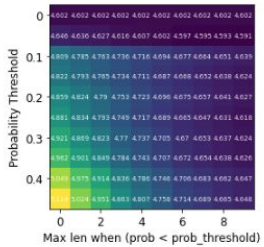
- 같은 모델에서 예측 답안의 최대 토큰 수 8일 경우, 12인 경우 편집거리 평균을 비교
- 각 답안 최대 길이와 확률 임계값에 따른 평균 편집거리를 구해 시각화하여,
  가장 낮은 편집거리를 가진 토큰 수와 확률 임계값으로 답안 후처리 실시
- 우측 예시는 KoBigBird 모델에서의 실험값으로,
  최대 토큰 수 12일 때
  - 최대 답안 길이 : 9
  - 확률 임계값: 0.05
  값에서 가장 낮은 편집거리 도출



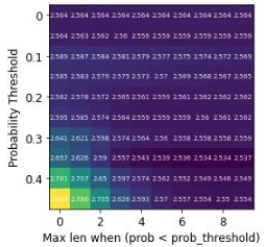




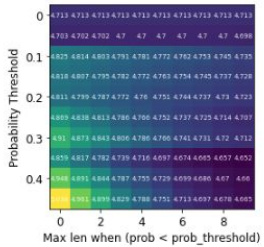
최대 토큰 수 8일 때 편집 거리 평균
(Best = 2.534)



최대 토큰 수 8일 때 편집 거리 표준 편차



최대 토큰 수 12일 때 편집 거리 평균
(Best = 2.537)



최대 토큰 수 12일 때 편집 거리 표준 편차

4.2.1 모델 개선 과정

KoELECTRA **(Kaggle) 14.519 → 3.086**

1. **베이스라인 설정에 KoELECTRA 모델 적용**

2. **토크나이저 변경 + 후처리 적용(길이 >20 인 경우 답안 공란처리)**

3. **베이스라인 학습 코드 → 커스텀 코드로 변경**
   - 베이스 라인 코드 내, model.train(), model.eval()의 부재로 인한 evaluation 단계에서 사용하지 않아도 될 batchnorm layer 혹은 dropout layer 과 같은 요소의 영향을 받은 loss를 전달, 효율적인 학습 저해
   - 따라서 KoELECTRA의 성능을 효과적으로 구현하기 위해 custom code를 작성하여 베이스라인 코드의 한계점을 극복

4. **확률로 자르기, token 길이 제한**
   - 예측한 정답의 확률과 토큰 길이 별 편집거리의 변화 기반, 정답확률 0.3 이하, token길이 8 이상인 답을 정답에서 제외하고 공백으로 처리

4.2.1 모델 개선 과정

KoBigBird (Kaggle) 2.774 → 2.473

1. **확률로 자르기, token 길이 제한**
   - 예측한 정답의 확률과 토큰 길이 별 편집거리의 변화 기반, 정답확률 0.3 이하, token길이 8 이상인 답을 정답에서 제외하고 공백으로 처리

2. **짧은 답만 학습하기**
   - 편집거리를 고려해 짧은 정답을 추론하도록, 입력 데이터에 여러 답이 있을 때 짧은 답만 선택해서 학습 데이터로 만들기

3. **KoBigBird 2048, 4096**
   - 2048 valid loss: 2.734 > 4096 valid loss : 2.703로
   - max_length가 더 길때 미세하게 성능이 개선되는 것 확인

4. **Token 길이 및 확률값에 따른 후처리**
   - 답안 최대 토큰 갯수 8->12개로 확장
   - 답안 확률 0.05 이하일 경우 답안을 5글자 이내로 자름

5. 자체 평가 및 보완사항

Pros

- 지난 프로젝트보다 자료 공유.아카이브가 잘 이루어 짐
- 자료조사, 전처리, 후처리, 모델 학습의 분업이 잘 이루어짐
- 비교적 좋은 점수 기록

Cons

- 최초 점수에 비해 낮은 kaggle 점수 개선
- 두 모델에 대한 동일한 실험 조건 관리의 부족함
- 버전 관리 미흡, 다음 프로젝트는 github을 활용할 예정

6. 참고 문헌

- **koBigBird github** https://github.com/monologg/KoBigBird
- **koELECTRA github** https://github.com/monologg/KoELECTRA
- **MeCab** https://bitbucket.org/eunjeon/mecab-ko-dic/src/master/
- **AI-Hub 뉴스 기사 기계독해** https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=577